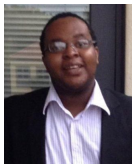
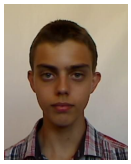
Word Predictor Using N-gram Models

Pierre Petitbon Rendani Mbuvha Pierre Godard Sudhanshu Mittal

petitbon@kth.se rendani@kth.se pgordard@kth.se smittal@kth.se


Abstract

Word Prediction is an important NLP problem where we predict the correct word given an incomplete sentence. The essence of this project is to take a corpus or create a domain specific corpus of text from various sources, analyze the text and build a predictive model to suggest the next most likely word given the last one/two/three words. We also perform word completion along with word prediction. Initially, we use the common approach to handle such problems by training and applying word n-gram model. The system needs to append with smoothing techniques for bigrams and trigrams which are not present in the vocabulary. Further steps include POS tagging, and back -off method to make the system syntactically correct and robust to word sense disambiguation. This system includes both word completion as well as word prediction. Lastly, various modules of the system are evaluated and compared using the measures named Keystroke Saving.

1. Introduction

Word prediction is one of the fundamental problems of Natural Language processing (NLP) **[9]**. This is the problem of guessing which words are likely to follow a given sequence of words. The importance of word prediction is primarily due to an increase in the number of software systems that require some form of text based user input. Systems of this nature include instant messaging, word processors and web pages to mention but a few. Efficient word prediction algorithms embedded in such systems result in positive user experiences as this can reduce the amount of time and cognitive effort spent on typing(saved keystrokes).

N-grams are a well known class of language models which rely on learning the conditional probability of a certain n-th word given the preceding n-1 words from a training dataset (corpora). The main drawback of N-gram models is that the resulting probability distribution

can be quite sparse due to the specific context of the training corpus[8]. Techniques such as smoothing and backoff can be used to redistribute the probability mass such that resulting model is more generalisable and with a reduced bias. Another issue is that most natural languages often have words that are ambiguous and have a different meanings depending on the context in which they are used. This is corrected by allowing for Parts of Speech Tagging (POS) to be used in combination with the n-gram language model.

In this work , we discuss and evaluate the various types of n-gram models for word prediction; we also evaluate the effect of different smoothing techniques and POS tagging on model prediction performance. This evaluation is based on the number of keystrokes saved.

1.1. Contributions

We use NLTK python library to create the word prediction and word completion system. We use collocation class from the library to search for all the n-grams. We implemented ourselves the basic n-gram scoring techniques based on trigrams, bigrams and unigrams. We analyse our method over three different smoothing techniques, namely- Laplace, Good-Turing and Kneser-Ney to find the best smoothing technique. Further, we include POS tagging technique by implementing the linear model combination technique to further improve the result by putting grammar constraints using trigram model of Part-of-Speech tags.

2. Related work

A large number of techniques have been proposed for word prediction in the past few years. Most of the existing work on word-prediction systems use statistical prediction algorithms. The statistical information used in word prediction system is mostly n-gram language models. A major drawback is attached to just using unigrams and bigrams is that they do not consider the syntactic and semantic structure of the sentence. The predictors have achieved a performance ranging from 37% -55% depending on the type of the text.

The report in **[8]** provides a well-documented report describing the process and decisions used to develop a predictive text model. It provides a detailed analysis mentioning all the key considerations encountered during the implementation. This work follows the techniques discussed in this report along with ideas taken from well-defined surveys provided in **[2]** and **[3]**. In **[2],** Fazly presents a comprehensive review of prior related work in word prediction. Fazly also presents a collection of experiments on word prediction. The implemented and evaluated algorithms **[2]** were based on word unigrams and bigrams, and based on syntactic features like POS tags in the syntactic predictors, and combination. **[3]** Provides a large scale survey on word prediction system. This work helps us understand various possible techniques that can introduced and examined.

As far as new methods are concerned, a great amount of machine learning methods are used based on feature extraction and selection techniques. In **[4],** the proposed system uses a rich set of features that combine the information available in the local context with shallow parsing information. **[6]** Compares a Simple Recurrent Network (SRN) to N-grams in text prediction tasks using guess rank and Entropy as measures of performance.

In **[5]** the authors present a comprehensive introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. This provides in depth insight in

fundamental NLP concepts such as N-grams, Smoothing, Discounting and Back-off which form the basis of our work.

3. My method

4.1 System Pipeline

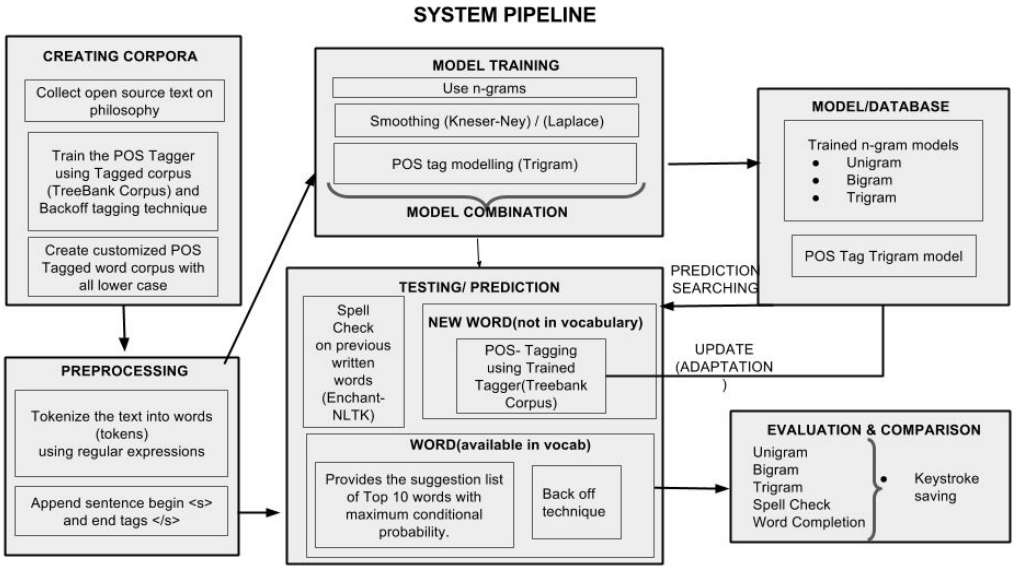


Fig1. This figure shows the complete pipeline of the word predictor system.

Now we describe all the modules used in the pipeline and also discuss about their implementation using NLTK library. Module mainly includes creating our customized corpora, tokenization of the text, ngram model, smoothing techniques, and POS tagging technique.

4.1.1 Creating Philosophy Corpora

We create a new philosophy corpora based of 60 philosophy texts and essays. Creating a customized corpora includes 3 steps. According to **[8],** the corpora words should not be case-sensitive. Although important for spelling correction and part of speech analysis, the words themselves- not their cases - are important for prediction. We convert the whole corpus to lower case. Although there is not more than 1% improvement in keystrokes saving by converting all the corpora into lowercase, but it helps in disambiguation of the same word appearing at different positions, thus treating them as equal.

The 3 steps involved in creating a tagged customized corpora includes.

- **Learning the tagger model:** It is done using some tagged corpus. In our case we use treebank corpus to train the tagger model. We use backoff training to learn tags, where tags are learnt using trigrams of tags, followed by bigrams of tags if trigrams are not found and so on.

- **Reading new Corpus:** Reading the new corpus includes tokenization of the corpus from paragraphs to sentences and further sentences to word tokens. We use self defined regular expression tokenizer for this purpose here.
- **Tagging the new Corpus:** The final step is to tag the new custom corpus based on the model learnt from the tagged corpus in the first step. All the tags along with the words are stored in a list of tuples. Here while storing the words, we convert new tagged words in lowercase for the reason as explained above.

4.1.2 Tokenization:

Tokenization is the process of splitting a string into a list of pieces or tokens. We tokenize sentences into words(tokens) using our own engineered regular expressions. Below is the list of the all tokenizer features that are used to creating separate tokens.

Features:

- Spaces between the words(word word))
- Abbreviations e.g. U.S.A
- Words with hyphens (word-word)
- Currency + number ($1.2)
- Decimal Numbers
- Commas and others punctuations are eliminated

A typical example of the tokenizer is given below:
sentence = "I am doing-well, Good muffins cost as much as $2.3 in U.S.A. "
tokens = ['I', 'am', 'doing-well', '.', 'Good', 'muffins', 'cost', 'as', 'much', 'as', '$2.3', 'U.S.A.']

We compare the results of the trigram model with the following tokenizers based of the measure - **Keystrokes Saving per Word**

- WordPunkt Tokenizer (counts all punctuations as tokens, thus also dividing words like **“can’t”** into three tokens- **‘can’, ‘ ` ’, ‘t’**)
- Regular Expression Tokenizer

Tokenizer	Keystroke Saving/ word
WordPunkt Tokenizer	2.919
Regular Expression Tokenizer	2.938
White Space Tokenizer	2.919

There is improvement of 2% in Keystroke saving per word using our customized tokenizer from default WordPunkt Tokenizer to our Regular Expression Tokenizer.

4.1.3 Spell correction

Spell correction is an important module used while testing the system. The text typed by the user should be correct to ensure that the word prediction algorithm works efficiently. Though spell correction is another application of n-grams model, but we use it as a preprocessing tool in our system. We use a spell correction API- **Enchant**. It is efficiently able to reduce the errors like repeating characters, letter missing inside the word and other minor spelling issues. Each time a word is typed, a method in the system will check Enchant to see whether the word is valid.

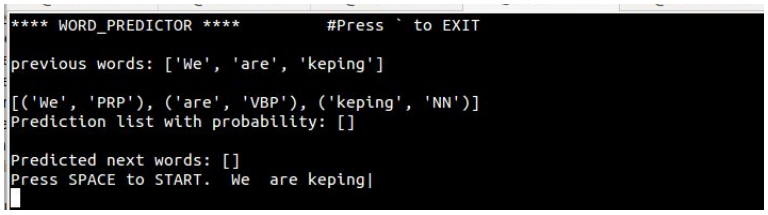


Fig. 2 The figure shows the user interface when user types in a incorrect word in the system ‘keping’

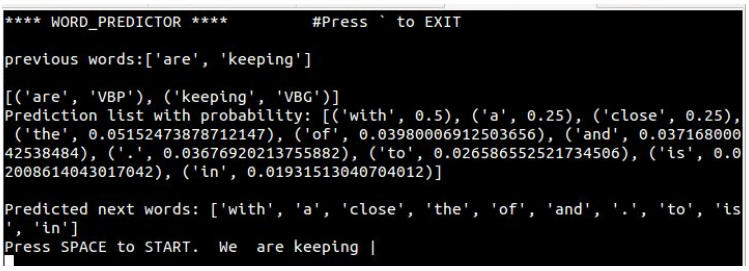


Fig.3 Here the mistyped word ‘keping’ becomes ‘keeping’, thus predicting the next word properly.

4.1.4 N Grams

N-Gram models were used for word prediction tasks described in this report. N-grams are a well known models of word sequences which are built on the simplifying assumption that the n-th word can be predicted based on the knowledge of the preceding n-1 words. Thus this is an N-1 order Markov Model. The N-Gram model approximates the probability of the n-th word as follows**[5]**:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

N-gram models are trained by counting the number of occurrences of a particular “n-gram” in the training corpus and normalising by the total of occurrences of the n-1 gram**[5]**.

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} W_n)}{C(w_{n-N+1}^{n-1})}$$

4.1.5 Smoothing

The major drawback with unsmoothed N-gram models it that any corpora is limited, and as result some perfectly acceptable word sequences will be missing from for it [5]. Thus this

will lead to a sparse distribution where there is a large number of "zero probability n-grams". Thus to reduce the bias in the training set one must adjust the probability mass from highly represented n-grams and redistribute it to the sparsely represented n-grams. The following smoothing techniques were used to achieve this:

4.1.5.1 Laplace Smoothing

This is a simple technique which adds one to all the n-gram counts in the corpus. The resulting probability mass is represented by:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{1+C(w_{n-N+1}^{n-1}W_n)}{\sum_{w_i} 1+C(w_{n-N+1}^{n-1})}$$

4.1.5.2 Good-Turing Smoothing

Good-Turing Smoothing re-assigns the probability mass of n-grams that occur c times in the training corpora to all n-grams that occur c-1 times. Good-Turing smoothing re-estimates the count of N-grams as follows:

$$c^* = (c+1)\frac{N_{c+1}}{N_C}$$

Where N_c *and* N_{c+1} the numbers of N-grams with counts c and c+1 in the original corpus respectively.The probability mass is therefore:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C^*(w_{n-N+1}^{n-1}W_n)}{\sum_{w_i} C^*(w_{n-N+1}^{n-1})}$$

4.1.5.3 Kneser-Ney Smoothing

This is the most commonly used smoothing technique. This adjusts the counts of n-grams by subtracting a fixed value from the count based on the preceding words. This is an interpolation technique which make use of high-order and low-order n-grams. For example, if the previous words frequency is near zero, the low-order model will carry more weight. Inversely, when the higher-order model match strongly, the lower-order model will be less weighted.

4.1.6 POS Tagging

POS tagging is essential for grammar analysis and word sense disambiguation. We are using universal tagset, which is a simplified and condensed tagset composed of 12 POS tags. This is supported by the NLTK Library. Firstly, the tagger is trained using the trained corpus(Brown corpus). NLTK uses a 'NgramTagger' class for tagging the new customized dataset, philosophy text in our case. Then any new philosophy text is tagged with upto 90% accuracy.

We can qualitatively compare the results after adding the POS tagging module. Below figure shows one simple example showing the improvement in prediction.

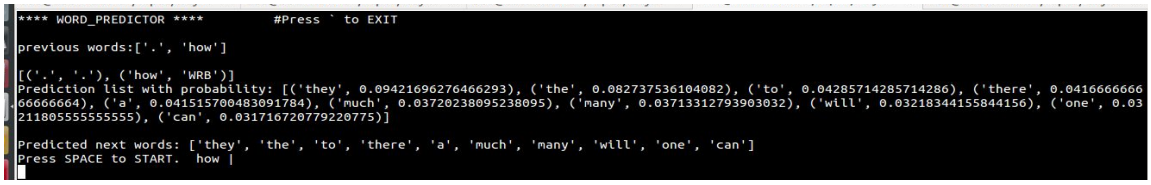


Fig.4 This figure shows the result produced using Trigram Model without POS Tagging.

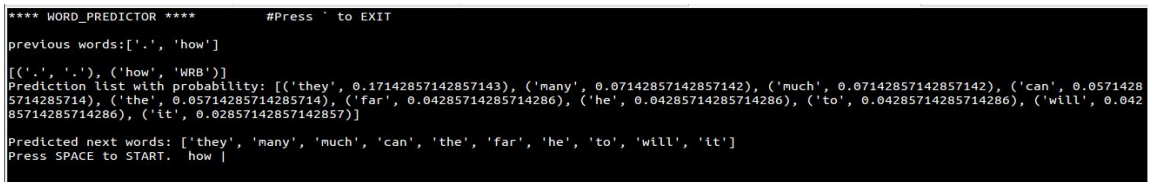


Fig.5 This Figure shows the result produced using Trigram Model with POS Tagging.

4.1.7 Model Combination

As N increases in the n-gram model, the power to express also increases, but the ability to estimate accurate parameters decreases from sparse datasets. A general approach is to combine the results of multiple N-gram models of increasing complexity (i.e. increasing N). Linearly combine estimates of N-gram models of increasing order. We are here combining a bigram model for words and a trigram model for tags.

The main idea of the linear combination approach is that the predictor first attempts to find the most likely part-of-speech tag for the current position according to the tags of the two previous words, i.e. maximizing $P(t_{cw}|t_{pw}, t_{ppw})$. Then to find words that have the highest probability of being in the current position according to POS tag. It is combined with probability of word given the previous n-1 words using n-gram model.

$$\alpha * P(w_i| pw) + (1-\alpha) * P(w_i|t_{cw}) * P(t_{cw}|t_{pw}, t_{ppw})$$

, where α is a coefficient between 0 and 1, w_i is the current word, pw is the previous word, t_{cw} is the tag for current word, t_{pw} is the tag for previous word and t_{ppw} is the tag for previous to the previous word.

4.2 Implementation

We use nltk collocation class to find all the n-gram collocations. A collocation is a sequence of words that occur together often.

4.2.1 N-grams

The n-gram are built with the ngrams function from nltk that find all the n-gram from a sequence of tokens. Then, all the different n-grams are counted and stored in a frequency distribution object, that basically is a dictionary where the n-gram is the key and the counts of

this bigram is the value (also called frequency in nltk). The frequency distribution object is used in the computation of all the following algorithms.

4.2.2 Collocation Finder

In order to implement the different NLP algorithm, we are using the collocation finder class from nltk. This gives our code a structure as all the algorithm will be implemented through this class. A collocation finder class contains a method for scoring n-grams. Another possibility would be to use the probability distribution classes, but the use of the collocation class was retained.

4.2.3 Basic n-gram techniques implementations

We implemented ourselves basic n-gram scoring techniques respectively based on trigrams, bigrams and unigrams. Trigrams and bigrams based techniques simply computes the conditional probability given the n-1 previous word. The unigrams based one basically computes the probability of seeing the word in the text.

4.2.4 Smoothing techniques implementations

We actually use three different smoothing techniques in our code : Laplace, Good-Turing and Kneser-Ney.

Laplace is used to smooth the bigram probabilities. So we want to compare this technique with a basic implementation based on bigrams. Good-Turing is also based on bigram. Nevertheless, the literature warn us about its poor performances when used alone (it is better when used as coefficient in more complex algorithm, like Katz backoff algorithm for example). That's why we won't compare it to other techniques. Kneser-Ney technique is built with trigrams as higher-order model and bigrams as lower-order model. As a smoothing technique, we want to compare it to Laplace smoothing technique. These three smoothing techniques implementations we are using are from nltk.

4.2.5 Model combination technique (using POS tagging)

The model combination technique, as seen before, is a linear combination of a bigram model based on the words and a trigram model based on the tags. In order to implement this technique, we have to store the frequency of all the bigrams and unigrams of words, and all the trigrams, bigrams and unigrams of tags. The conditional probability of the current word given its tag is computed using a frequency distribution object whose the key is a tuple containing the current word and his associated tag. So the model combination formula:

$$\alpha * P(w_i | pw) + (1-\alpha) * P(w_i|t_{cw}) * P(t_{cw}|t_{pw}, t_{ppw})$$

can be rewritten like these

$$\alpha * \frac{C(W_{n-1}W_n)}{C(W_{n-1})} + (1-\alpha) * \frac{C(W_{n-1}t_n)}{C(t_n)} * \frac{C(t_{n-2}t_{n-1}t_n)}{C(t_{n-2}t_{n-1})},$$

,where α is a coefficient to be fixed, W_n is the current word, W_{n-1} is the previous word, t_n is the current tag, t_{n-1} is the previous word tag and t_{n-2} is the tag for the previous word to the previous word. $C(x)$ is the count of the x n-gram, stored in the corresponding frequency distribution object.

Note that before computing the linear combination, we have to found the best tag for the current word by maximizing $P(w_i|t_{cw})$ * $P(t_{cw}|t_{pw}, t_{ppw})$. Moreover the α coefficient have been fixed so that it optimizes the keystroke saving of the technique.

5. Experimental results

For the experimentation and evaluation of our system, we check our system on text from different authors. Firstly, we divide our philosophy corpora into training and validation sets. For testing, we randomly selected a trigram from the validation set, and simulate the typing procedure of the third word to evaluate the system in terms of Keystroke Saving. Later, we compare different features of the system with their effect on the performance of the system.

5.1 Experimental User Interface(Live Demo)

This method of word prediction is supposed to reduce the cognitive load on their users. In a word-prediction system, visually searching the list of suggestions and deciding on whether to select the word from a long list is not feasible. Thus, we focus on keeping the number of suggestions to limited of 10 words in decreasing order of prediction and increase the keystroke saving as much as possible.

In our system, when the user presses the " "(space key), the list of predictions appear as a list of 10 words. On the keystroke from 0-9, the found predicted work is appended to the sentence and user can continue to write further. If the word is not found in the suggestion list then the user continues to write the next word. At each keystroke from the user, the possible completion of that word is also predicted thus helping in completing the word. The user interface is shown in the screenshots given below.

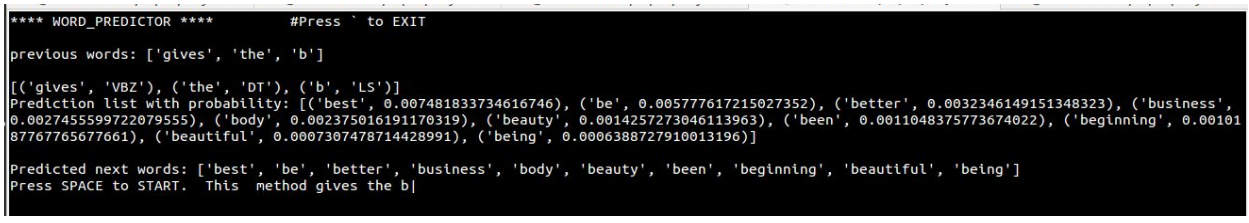


Fig.6 The figure displays the User Interface. As user types the word, a list of Top 10 predicted words appears with decreasing probability.

6. Evaluation and Comparison

6.1 Model Evaluation

Measure of Performance **:** KeyStroke Saving (why?)

Because the keystroke saving is the goal of all word prediction system, it is a very commonly used measure to evaluate and compare systems. It measures the percentage reduction in key pressed compared to a classic character-by-character typing.

6.2 Importance of Different Modules

We perform experimentation on different modules of the system. Since, the results are performed on random test cases, the overall value is each experiment might differ. Therefore, the improvement in percentage of performance is more important to make any conclusion about different models.

6.2.1 Word Completion

Below we show the improvement in performance of the system with respect to keystrokes saved per word when word completion module is included in the system.

MODEL	KeyStroke Saving / Word
Trigram Prediction without Word Completion	1.075
Trigram Prediction with Word Completion	2.737

Word completion is an important part of this application of word prediction. By adding the feature of word completion there is improvement of 150% in Keystroke Saving.

6.2.2 Simple N-gram Models

We run the n-gram models including uni-gram, bi-gram, tri-gram only. We observe an increase in performance from unigram to bigram. But the performance from decreases from bi-gram to tri-gram. Although tri-gram has higher performance than unigrams, but perform worse than bigrams. This experiment is carried over 300 randomly selected test cases.

MODEL	KeyStroke Saving / Word	%Keystrokes Saved
Unigram Model	1.28	34.22%
Bigram Model	1.67	44.74%
Trigram Model	1.24	33.15%

6.2.3 Comparison between different Smoothing Techniques

We experiment on different well known smoothing techniques with the simple tri-gram model. We observe the best performance in case of trigram model used with Kneser Ney smoothing. This comparison is also based of 300 selected test cases also used above. Trigram model used with Kneser Ney smoothing saves up to 49% of keystrokes.

MODEL	Keystroke Saving/ Word	% keystroke saving
Tri gram model without smoothing	1.24	33.15 %
Trigram model with Laplace Smoothing	1.77	47.41 %
Trigram model with Kneser Ney Smoothing	1.81	48.39 %

6.2.4 Bigram model with Trigram POS Tagging

After using different simple n-gram models, we get best performance for the bigram model. Here we combine the bigram model with the trigram POS tagging model to get the best result. It is a linear combination of bigram model and trigram POS tagger as explained in the methodology section.

MODEL	KeyStroke Saving/Word	Keystroke saving
Bigram System with POS tagging	1.8	26.8%
Bigram System without POS Tagging	1.0	14.9%

This test is performed on a different smaller tagged corpus because of limited computation resources(only using 8% of the corpus files used in above experiments), therefore it cannot be directly compared with above modules. Here, we separately compare bigram model with and without POS tagging. In the 'user_interface.py' code, which is used for live demo uses this combined model of POS tagging and bi-grams by default.

7. Summary and Conclusions

We test various modules for better word prediction and word completion system to save maximum possible keystrokes. We consider different n-gram models with limited available computational resources. We find bigram model as the best simple n-gram model from unigram, bigram and trigram. This gives upto an keystroke saving percentage of 45%. Further different smoothing techniques are explored and analysed with the n-gram models. Kneser Ney smoothing is found to be the best one amongst them with increase in keystroke saving upto 49%. To put grammar constraints and word sense disambiguation into effect by using Trigram POS tagger model combined with the bigram model to get better language model with an improvement in keystroke saving.

8. Contributions

We, the members of project group 62, unanimously declare that we have all equally contributed toward the completion of this project.

References

[1] Stuart J. Russell and Peter Norvig. 2003. Artificial Intelligence: A Modern Approach (2 Ed.). Pearson Education.

[2] Fazly A., "The Use of Syntax in Word Completion Utilities," Master Thesis, University of Toronto, Canada, 2002.

[3] Garay-Vitoria, Nestor, and Julio Abascal. "Text prediction systems: a survey."*Universal Access in the Information Society* 4.3 (2006): 188-203.

[4] Even-Zohar, Yair, and Dan Roth. "A classification approach to word prediction. *"Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Association for Computational Linguistics, 2000.

[5] Daniel Jurafsky and James H. Martin. 2000. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

[6] Paul Rodriguez. 2003. Comparing Simple Recurrent Networks and n-Grams in a Large Corpus

[7] Masood Ghayoomi and Ehsan Daroodi. "A POS-based Word Prediction System for the Persian

[8] Gerald R. Gendron, Jr, " Natural Language Processing: A Model to Predict a Sequence of Words"

[9] Hisham Al-Mubaid " A Learning-Classification Based Approach for Word Prediction "